Science of Computer Programming 12 (1989) 75-81
North-Holland

SMALL PROGRAMMING EXERCISES 23

M. REM

Department of Mathematics and Computing Science, Eindhoven University of Technology, Eindhoven, The Netherlands

Each integer array $X(i: 0 \leq i < N)$ of which all elements are within its domain $[0 \,..\, N)$ represents a function $X : [0 \,..\, N) \rightarrow [0 \,..\, N)$. Swapping array elements will, in general, cause the function thus represented to change. A surprising property, however, is that by just swapping array elements every function can be made idempotent, i.e. can be made to satisfy $X(X(i)) = X(i)$ for all $0 \leq i < N$. The problem of Exercise 53 is to program these swaps. The reader may appreciate the hint that a very simple linear program with just one auxiliary variable can do the job. I owe this exercise to A. Kaldewaij.

Consider the problem of deleting multiple values in an array of size K. This can obviously be done in $O(K \log K)$ time, for instance by first sorting the array. If all elements are taken from some finite set of N elements the introduction of a boolean array of size N allows an $O(K + N)$ solution. Consider now the problem of deleting multiple arcs in a directed graph of N vertices and M arcs. The successors of each vertex are given in an array segment. From each such segment the multiple values have to be deleted. Doing that by repeatedly employing the $O(K \log K)$ strategy will yield a solution that is—due to the logarithmic factor—worse than linear in M. Based on the $O(K + N)$ strategy, however, one can obtain an $O(M + N)$ algorithm. Exercise 54 asks the reader to find such a solution.

Exercise 53: Idempotence swap

We have to find a solution for S, whose only operations on X are swaps, in

```
|[ N: int; {N ≥ 0}
  |[ X(i: 0 ≤ i < N): array of int;
    {(Ai: 0 ≤ i < N: 0 ≤ X(i) < N)}
    S
    {(Ai: 0 ≤ i < N: X(X(i)) = X(i))}
  ]|
]|
```

Exercise 54: Multiple-arcs reduction

Directed graph G with N vertices, numbered 0 through $N-1$, is represented in arrays b and e by means of successor lists: array segment $e(i: b(j) \leq i < b(j+1))$

lists the successors of vertex j. In contrast to the standard representation, this list may contain multiple copies of vertices. We denote the fact that G is thus represented by MSUC(G, b, e). It is the purpose of this exercise to represent G in arrays $b0$ and $e0$ without multiple successors:

```
|[ N,M: int; {N ≥ 1 ∧ M ≥ 0}
  b(j: 0≤j≤N), e(i: 0≤i<M): array of int;
  {MSUC(G, b, e)}
  |[b0(j: 0≤j≤N), e0(i: 0≤i<M): array of int;
    S
    {SUC(G, b0, e0(i: 0≤i<b0(N)))}
  ]|
]|
```

Solution of Exercise 51 (supersource)

The exercise is to find a statement list S such that

```
|[ N: int; {N ≥ 1}
  E(i,j: 0≤i<N ∧ 0≤j<N): array of bool;
  |[b: bool; c: int;
    S
    {(Ei: 0≤i<N: SS(i)) ⇒ SS(c)
     ∧ b ≡ SS(c)}
  ]|
]|
```

where, for $0 \leq i < N$,

$$\mathrm{SS}(i) \equiv (\mathbf{A}j: 0 \leq j < N \wedge j \neq i: E(i,j) \wedge \neg E(j,i))$$

This problem was called The Celebrity Problem in [1].

Once the first conjunct

$$R: \quad (\mathbf{E}i: 0 \leq i < N: \mathrm{SS}(i)) \Rightarrow \mathrm{SS}(c)$$

of the postcondition has been established, it is simply a matter of applying the definition of SS(c) to establish its second conjunct $b \equiv \mathrm{SS}(c)$ as well. We, consequently, focus on establishing R.

Notice that R is implied by

$$(\mathbf{A}i: 0 \leq i < N: \mathrm{SS}(i) \Rightarrow i = c)$$

We introduce variable d and generalize the condition above into the following invariant:

$$P: \quad 0 \leq c \leq d < N \wedge (\mathbf{A}i: 0 \leq i < N: \mathrm{SS}(i) \Rightarrow c \leq i \leq d)$$

Then $P \wedge c = d$ implies R. Invariant P may be initialized with $c,d = 0,N-1$. The bound function is $d-c$, and we consider for $c \neq d$ statements $d := d-1$ and $c := c+1$ as candidates to decrease the value of this function. In order that these statements maintain P they have $\neg\mathrm{SS}(d)$ and $\neg\mathrm{SS}(c)$ as their respective preconditions. By definition, $\neg\mathrm{SS}(d)$ is equivalent to

$$(\mathbf{E}j\colon 0 \leq j < N \wedge j \neq d\colon \neg E(d,j) \vee E(j,d))$$

For $c \neq d$ we conclude $E(c,d) \Rightarrow \neg\mathrm{SS}(d)$ and, similarly, $\neg E(c,d) \Rightarrow \neg\mathrm{SS}(c)$. Thus we find two complementary guards, $E(c,d)$ and $\neg E(c,d)$, for our candidate statements.

This concludes the derivation. The resulting program text is

```
S:   |[d: int; c,d := 0,N-1
     ; do c ≠ d → if E(c, d) → d := d-1
                  □ ¬E(c, d) → c := c+1
                  fi
       od
     ]|  {R}
   ; |[j: int; j,b := 0,true
     ; do j ≠ N ∧ b → b := (j = c) ∨ (E(c,j) ∧ ¬E(j, c))
                    ; j := j+1
       od
     ]|  {R ∧ b ≡ SS(c)}
```

Although matrix E has N^2 elements, the execution of S is linear in N.

Solution of Exercise 52 (series-parallel graphs)

The reader is advised to consult *Small Programming Exercises 22* for a definition of SP graphs. We have to find a solution for S in

```
|[N, T, U: int; {0 ≤ T < N ∧ 0 ≤ U < N ∧ T ≠ U}
  E(i,j: 0 ≤ i < N ∧ 0 ≤ j < N): array of int;
  {(Ai: 0 ≤ i < N: E(i, i) = 0 ∧ (Aj: 0 ≤ j < N: 0 ≤ E(i,j) = E(j, i)))}
  |[b: bool;
    S
    {b ≡ (graph G with terminals T and U is series-parallel)}
  ]|
]|
```

We call a vertex that is not a terminal a nonterminal. Two vertices are called neighbours if they are connected by at least one edge. The multiplicity of its edges

is immaterial for the question whether a graph is SP. We shall, therefore, in the following analysis confine our attention to graphs that do not have multiple edges.

SP graphs of at least three vertices contain at least one nonterminal that has exactly two neighbours. This property can easily be verified by induction over the construction rules of SP graphs. It holds in the base case, since that graph has only two vertices. Assume the property to hold for the two graphs in a composition. If at least one of these contains a nonterminal that has two neighbours, then so will this vertex in the composite. If, on the other hand, both graphs contain exactly two vertices the parallel composition will still have two vertices; in the serial composition, which then has three vertices, the vertex by which the graphs are joined will have the two new terminals as its only neighbours.

This property suggests a solution method that I heard from W. Kloosterhuis. The crux of the method is to decrease the number of vertices in the graph under invariance of its SP-ness, i.e. in such a way that the reduced graph is SP if and only if the original one was SP. We accomplish this reduction by continually selecting a nonterminal with two neighbours, removing that vertex, and connecting its two neighbours. A graph that does not contain a selectable vertex is SP if and only if it consists of two connected vertices.

We show that reduction steps maintain SP-ness by (i) and (ii). Let $\mathrm{red}(G, x)$ be the graph that results after applying to G a reduction step with x as the selected vertex. Let $\{y, z\}$ be the set of neighbours of x.

(i) *G is SP implies* $\mathrm{red}(G, x)$ *is SP.* Repeat the construction of G, but leave out the composition with the base graph of vertices x and z, and replace in all succeeding steps vertex x by z. The SP graph that then results is $\mathrm{red}(G, x)$.

(ii) $\mathrm{red}(G, x)$ *is SP implies G is SP.* Consider a construction of $\mathrm{red}(G, x)$. It uses the base graph of vertices y and z. Serially compose this with a base graph of vertices z and x, and use the result instead. Thus we obtain a construction of graph G.

We can now give the structure of our program. Graph $\langle W, F\rangle$ is the one that is being reduced, with W as its vertex set and $F \subseteq W \times W$. We maintain the invariance of

$$\{T, U\} \subseteq W \subseteq \{0, 1, \ldots, N-1\}$$
$$\wedge\ (\langle W, F\rangle \text{ is SP}) \equiv (G \text{ is SP})$$

Set V is the set of selectable vertices in W:

$$V = \{i \in W \setminus \{T, U\} \mid (\mathbf{N}j: j \in W: (i, j) \in F) = 2\}$$

The program has the following structure:

$$W := \{0, 1, \ldots, N-1\}$$
$$; F := \{(i, j) \in W \times W \mid E(i, j) \geq 1\}$$
$$; V := \{i \in W \setminus \{T, U\} \mid (\mathbf{N}j: j \in W: E(i, j) \geq 1) \geq 2\}$$
$$; \textbf{do } V \neq \emptyset$$

$\rightarrow$ let $x \in V$; $V := V \backslash \{x\}$; $W := W \backslash \{x\}$

; let $(x, y) \in F \wedge (x, z) \in F \wedge y \neq z$

; $F := (F \backslash \{(y, x), (x, y), (z, x), (x, z)\}) \cup \{(y, z), (z, y)\}$

; **for** $i \in \{y, z\} \backslash \{T, U\}$

do if $(\mathbf{N}j: j \in W: (i, j) \in F) = 2 \rightarrow V := V \cup \{i\}$

$\square$ $(\mathbf{N}j: j \in W: (i, j) \in F) \neq 2 \rightarrow$ **skip**

fi

od

od

; $b := (W = \{T, U\} \wedge (T, U) \in F)$

To obtain the final program we have to choose representations for W, F, and V. Set W is enumerated in segment $w(i: 0 \leq i < nw)$ of integer array w. Variable nw then records the cardinality of W. Set F is represented by boolean array $f(i, j: 0 \leq i < N \wedge 0 \leq j < N)$:

$$(\mathbf{A}i, j: i \in W \wedge j \in W: f(i, j) \equiv (i, j) \in F)$$

In order to code the guards in the selection we introduce integer array $n(i: 0 \leq i < N)$ and let $n(i)$ be the number of neighbours of $w(i)$ in graph $\langle W, F \rangle$:

$$(\mathbf{A}i: 0 \leq i < nw: n(i) = (\mathbf{N}j: j \in W: (w(i), j) \in F))$$

We do not represent set V; we record only its cardinality in variable nn. Thus the initialization becomes

INIT: $nw, nn := N, 0$

; $|[i: int; i := 0$

; **do** $i \neq N$

$\rightarrow w: (i) = i$; $n: (i) = 0$

; $|[j: int; j := 0$

; **do** $j \neq N$

$\rightarrow f: (i, j) = (E(i, j) \geq 1)$

; **if** $f(i, j) \rightarrow n: (i) = n(i) + 1 \square \neg f(i, j) \rightarrow$ **skip fi**

; $j := j + 1$

od

$]|$

```
    ; if n(i) = 2 ∧ i ≠ T ∧ i ≠ U → nn := nn + 1
      □ n(i) ≠ 2 ∨ i = T ∨ i = U → skip
      fi
    ; i := i + 1
    od
  ]|
```

The execution time of INIT is quadratic in N.

Our solution for S may now be coded as

```
S:  |[nw,nn: int; w,n(j: 0 ≤ j < N): array of int;
     f(i,j: 0 ≤ i < N ∧ 0 ≤ j < N): array of bool;
     INIT
    ; do nn ≠ 0
     → |[i,j,x,y,z: int; i := 0
        ; do n(i) ≠ 2 ∨ w(i) = T ∨ w(i) = U → i := i + 1 od
        ; x := w(i); nw,nn := nw − 1,nn − 1
        ; w: swap(i, nw); n: swap(i, nw)
        ; i := 0; do ¬f(x, w(i)) → i := i + 1 od; y := w(i)
        ; j := i + 1; do ¬f(x, w(j)) → j := j + 1 od; z := w(j)
        ; if f(y, z)
          → n: (i) = n(i) − 1
            ; if n(i) = 2 ∧ w(i) ≠ T ∧ w(i) ≠ U → nn := nn + 1
              □ n(i) ≠ 2 ∨ w(i) = T ∨ w(i) = U → skip
              fi
            ; n: (j) = n(j) − 1
            ; if n(j) = 2 ∧ w(j) ≠ T ∧ w(j) ≠ U → nn := nn + 1
              □ n(j) ≠ 2 ∨ w(j) = T ∨ w(j) = U → skip
              fi
          □ ¬f(y, z)
          → f: (y, z) = true; f: (z, y) = true
          fi
        ]|
    od
    ; b := (nw = 2 ∧ f(T, U))
  ]|
```

We have designed an $O(N^2)$ program for determining whether a graph is series-parallel. This is, of course, the best one can achieve for graphs that are given as incidence matrices.

Reference

[1] U. Manber, Using induction to design algorithms, *Comm. ACM* **31** (1988) 1300-1313.